Exp No: 9

Srinidhi.V
18CL02

Develop Any Rule Based System for an application

Water Jug Problem.

Aim:

To implement the water jug problem using bfs and state-space (bfs).

Algorithm:

* Start
* Read the capacities of jug1, jug2 and final capacity of water needed.
* Call the function water-jug-bfs().
* In function, water-jug-bfs(), add the target values & if the target list is in current list, break the loop in function.
* Fill both the jugs and empty both the jugs based on the rule.
* If at any instant, the x gallon jug becomes empty, fill it with water.
* If at any instant, the y gallon jug becomes empty, fill it with water.
* Do steps 5, 6 & 7 till any of the jugs among the x gallon and y gallon jugs contains exactly z litres of water using rules.
* Stop

Problem:

We have 2 water jugs, one measure x gallon & other one measure y gallon. But there is no measuring label on either of these jugs. (ie) we can't know the exact amount filled in the jug.

i) There is infinite amount of water supply

ii) We can empty / fill the jugs completely.

iii) We can transfer water from 1 jug to another.

→ Fill 2G of water into anyone of this jug.

Using BFS.

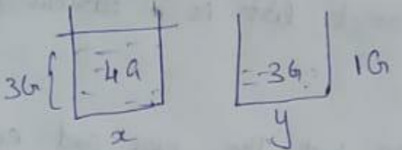
$x=0$ $y=0$

$x=0$ $y=3$

$x=3$ $y=0$

$x=3$ $y=3$

$x=4$ $y=2$



Using State-Space (BFS):

The state space for this problem can be described as the set of ordered pairs of integers (x, y)

$x \rightarrow$ The quantity of water in 4G Jug.

$y \rightarrow$ The quantity of water in 3G Jug.

State space = (0,0)

Goal state = (2,0)

Production Rule:

Rule	State	Process
1	$(x, y \mid x < 4)$	$(4, y)$ {Fill The 4G Jug}
2	$(x, y \mid y < 3)$	$(x, 3)$ {Fill The 3G Jug}
3	$(x, y \mid x > 0)$	$(0, y)$ {Empty The 4-gallon jug}
4	$(x, y \mid y > 0)$	$(x, 0)$ {Empty 3G Jug}
5	$(x, y \mid x+y \geq 4$ & $y > 0)$	$(4, y-(4-x))$ {Pour water from 3 to 4 gallon Jug until 4 gallon jug is full}
6	$(x, y \mid x+y \geq 3, x > 0)$	$(x-(3-y), 3)$ {Pour water from 4 to the other 3G Jug until 3G is full}
7	$(x, y \mid x+y \leq 4$ & $y > 0)$	$(x+y, 0)$ {Pour all the water from 3 to 4G Jug}
8	$(x, y \mid x+y \leq 4$	$(0, x+y)$ {Pour all the water from 4 to 3G Jug}.

Gallons in 4 G Jug (x)	Gallon in 3G Jug (y)	Rule Applied.
0	0	1
4	0	6
1	3	4
1	0	8
0	1	1
4	1	6
2	3	4
2	0	Goal achieved.

Program:

Breadth First Search:

```
from collections import deque

def printPath(target, parent):
    res = []
    curr = target
    src = (-1, -1)

    while(curr != src):
        res.append(curr)
        curr = parent[curr]

    print()
    while(len(res) != 0):
        print(res.pop(), end = " -> ")
    print("GOAL")

# Available operations:
#    1. Fill the jug
#    2. Empty the jug
#    3. Transfer jug contents

def water_jug_bfs(j1, j2, water):

    visited = set()                   # To hold the already visited nodes
    q = deque()                       # To hold the bfs queue
    parent = dict()                   # To store parent of any node
    q.append((0, 0))                  # initially we start with (0, 0) as the starting state
    visited.add((0, 0))
    parent[(0, 0)] = (-1, -1)         # the starting state has no parent
    isSolvable = False                # Sometimes problem cant be solved
    target = [(0, water), (water, 0)]   # required target state

    while(len(q) != 0):
```

```
curr = q.popleft();
if(curr in target):
    isSolvable = True
    break

curr_j1, curr_j2 = curr[0], curr[1]
possiblities = []

possiblities.append((j1, curr_j2))  # 1a) Fill jug1
possiblities.append((curr_j1, j2))  # 1b) Fill jug2

possiblities.append((0, curr_j2))   # 2a) Empty jug1
possiblities.append((curr_j1, 0))   # 2b) Empty jug2

# 3a) Jug-1 to Jug-2
# cant transfer when jug-1 is empty and jug-2 is already full
if(curr_j1 != 0 and curr_j2 != j2):
    total_water = curr_j1 + curr_j2
    # when total capacity is less than jug-2 capacity
    if(total_water <= j2):  possiblities.append((0, total_water))
    # when total capacity is greater than jug-2 capacity
    else:   possiblities.append((total_water-j2, j2))

# 3b) Jug-2 to Jug-1
# cant transfer when jug-2 is empty and jug-1 is already full
if(curr_j1 != j1 and curr_j2 != 0):
    total_water = curr_j1 + curr_j2
    # when total capacity is less than jug-1 capacity
    if(total_water <= j1):  possiblities.append((total_water, 0))
    # when total capacity is greater than jug-1 capacity
    else:   possiblities.append((j1, total_water-j1))

for poss in possiblities:
    if(poss not in visited):
        x, y = poss[0], poss[1]
        q.append((x, y))
        visited.add((x, y))
        parent[(x, y)] = curr
```

```
    if(isSolvable):
        printPath(curr, parent)
    else:
        print("Not possible to work with these inputs")


if __name__ == "__main__":

    jug1 = int(input("Enter jug 1 capacity : "))
    jug2 = int(input("Enter jug 2 capacity : "))
    water = int(input("Enter final capacity of water needed : "))
    water_jug_bfs(jug1, jug2, water)
```

Output:

```
Enter jug 1 capacity : 4

Enter jug 2 capacity : 3

Enter final capacity of water needed : 2

(0, 0) -> (0, 3) -> (3, 0) -> (3, 3) -> (4, 2) -> (0, 2) -> GOAL
```

Using State Space (BFS):

```
from collections import deque

def water_jug_bfs(j1, j2, water):

    visited = set()          # To hold the already visited nodes
    q = deque()              # To hold the bfs queue
    q.append((0, 0))         # initially we start with (0, 0) as the starting state
    print("\n", (0, 0))
    visited.add((0, 0))
    isSolvable = False       # Sometimes problem cant be solved
```

```
target = [(0, water), (water, 0)]   # required target state

while(len(q) != 0):

    size = len(q)
    print("\n\n *** \n")

    for _ in range(size):

        curr = q.popleft();
        if(curr in target):
            isSolvable = True
            break

        curr_j1, curr_j2 = curr[0], curr[1]
        possiblities = []

        possiblities.append((j1, curr_j2))  # 1a) Fill jug1
        possiblities.append((curr_j1, j2))  # 1b) Fill jug2

        possiblities.append((0, curr_j2))   # 2a) Empty jug1
        possiblities.append((curr_j1, 0))   # 2b) Empty jug2

        # 3a) Jug-1 to Jug-2
        # cant transfer when jug-1 is empty and jug-2 is already full
        if(curr_j1 != 0 and curr_j2 != j2):
            total_water = curr_j1 + curr_j2
            # when total capacity is less than jug-2 capacity
            if(total_water <= j2):  possiblities.append((0, total_water))
            # when total capacity is greater than jug-2 capacity
            else:   possiblities.append((total_water-j2, j2))

        # 3b) Jug-2 to Jug-1
        # cant transfer when jug-2 is empty and jug-1 is already full
        if(curr_j1 != j1 and curr_j2 != 0):
            total_water = curr_j1 + curr_j2
            # when total capacity is less than jug-1 capacity
            if(total_water <= j1):  possiblities.append((total_water, 0))
```

```
            # when total capacity is greater than jug-1 capacity
            else:   possiblities.append((j1, total_water-j1))

        for poss in possiblities:
            if(poss not in visited):
                x, y = poss[0], poss[1]
                q.append((x, y))
                print((x, y), end= " ")
                visited.add((x, y))

    if(isSolvable == False):
        print("Not possible to work with these inputs")


if __name__ == "__main__":

    jug1 = int(input("Enter jug 1 capacity : "))
    jug2 = int(input("Enter jug 2 capacity : "))
    water = int(input("Enter final capacity of water needed : "))
    water_jug_bfs(jug1, jug2, water)
```

Output:

```
Enter jug 1 capacity : 4

Enter jug 2 capacity : 3

Enter final capacity of water needed : 2

 (0, 0)


 ***

(4, 0) (0, 3)

 ***

(4, 3) (1, 3) (3, 0)

 ***

(1, 0) (3, 3)

 ***

(0, 1) (4, 2)

 ***

(4, 1) (0, 2)

 ***

(2, 3)

 ***

(2, 0)
```

Result:

Thus the rule based system (i.e) Water Jug Problem is implemented using bfs and State_space (bfs).